Intro to Java

Java Composition

- Java shares much of the same syntax with C/C++
  - Functional and algorithmic scopes are bounded with brackets
  - Primitives tend to be the same
    - ‘int’ is still ‘int’
    - ‘long’ is still ‘long’
    - ‘double’ is still ‘double’
    - ’bool’ is instead ’boolean’ but is close enough
  - Algorithmic conveniences are the same
    - ’if’, ‘switch’, ‘for’, ‘while’, ‘do while’ are all present and work identically to C
  - There are some small differences with familiar types
    - A ‘string’ is a ‘String’ in Java, and is an object rather than a type façade over a char array. (there is a String object in C++ too…)
- Methods (functions) are declared in a manner similar to C/C++, and may be void or value returning.

Java Keywords

- Scoping keywords are the same
  - 'private', 'protected', 'public', 'this' all do the same thing
- Some keywords for creating objects remain the same, some do not
  - 'class' remains
  - 'struct' does not exist in Java
  - 'abstract' and 'interface' are Java keywords for creating conditionally typed inheritance (more on this later)
  - 'final' replaces 'const'
  - 'static' remains more or less the same

Java Annotations

- An annotation is a means for marking classes, methods, variables, parameters and even packages with syntactic metadata that can be used as a compiler or runtime directive.
- Annotations are recognizable by an '@' symbol prefix.
- We will see several annotations throughout this course, but the most common that we will deal with will be the '@Override' annotation through our exploration of abstract classes and interfaces.

Intro to Object Oriented Programming (OOP)

What is an Object

- An object is a self contained entity that contains attributes or behavior (a class).
- Objects are distinguishable as either Smart or Dumb
  - Dumb objects are property bags meant to represent a domain component from a purely data perspective.
  - Smart objects not only contain properties but also behavioral program logic
- Objects communicate by ‘sending messages’ which is another way of saying ‘method calls’

Well Written Objects

- Have well defined boundaries
- Perform a set of finite activities
- Knows only about its data and any other objects that it needs to accomplish it's activities
- Is a discrete entity that only has dependencies on other objects to the degree that it needs to perform its tasks

Example: Person

- What attributes does a person have?
  - Name
  - Age
  - Height
  - Weight
  - Eye color
  - Gender

Example: Person (cont)

- What behaviors does a person have? Best represented as verbs describing actions that a person might take or that might provide data beyond the provided interface.
  - Walk
  - Breathe
  - Calculate BMI
  - Calculate Blood Pressure

Object State

- State is represented by the value of an Object's attributes at a moment in time.
- Using the Person example, state is defined by all of the Person specific attributes defined in the interface.
- An object's state may be checked by invoking accessors to all attributes in the interface.

Principles of OOP

- Three pillars of OOP that are regarded as primary principles
  - Encapsulation
  - Inheritance
  - Polymorphism

Encapsulation

- As the definition of a well-written object states, objects should be discrete (self-contained). This is the principle of Encapsulation (sometimes called 'Hiding').
- In brief, the principle of encapsulation dictates that an object should maintain a boundary between it's state and behaviors and the rest of the universe.
- Achieved through access modifiers;
- Question: public attributes, or private with accessors and mutators?

Inheritance

- In class based programming, it has been a practice to copy a class (or structure) and enhance or extend that class through the addition of attributes or behaviors. This results in repeated code.
- Inheritance allows for the enhancement or extension of an object into a new form without copying code, but rather by allowing that code to be passed down to inheriting classes. This is a Parent/Child relationship, also referred to as Base/Sub classing
- Behaviors or Attributes that need to be modified in Child classes may be overridden.
- Example: Person Inheritance

Polymorphism

- The principle of polymorphism is abstract (no pun intended)
- In brief: Polymorphism means that objects which belong to the same inheritance hierarchy and which share a common behavior might express the results of that behavior in different ways
- Example: Super Heroes

Simple Rules for Well-Defined Objects

Nouns

- Member variables on objects are meant to represent or quantify some value. In this way they are like nouns, and should be named as such.
  - int speed;
  - string name;
  - Booleans may be slightly different in that they contain a true/false (yes/no) value, and may be named with that in mind:

    boolean isCorrect;

Verbs

- Methods (functions) should be representative of some action or behavior, whether it is calculated or directive. In this way they are like verbs and should be named as such.
  - public void accelerate(int upperBound);
  - public int calculateTurnSpeed(int currentRate, int turnRadius);
  - Again, booleans are different in that they return the answer to a binary question, and the phrasing for those methods should continue to reflect that:

    public boolean isLegalSpeed(int currentSpeed);

Accessors and Mutators

- Accessors (Getters) and Mutators (Setters) follow the same convention as other methods in that they are still actions.
- Getters and Setters should never contain more logic than necessary to fulfill their base purpose (true of all functions).
- Getters and Setters should always have the same naming conventions
  - Setter method names are always prefixed by 'set' and followed by the name of the variable being set.
  - Getter method names are always prefixed by 'set' and followed by the name of the variable being set.

```
private int speed;

public void setSpeed(int value) {
    this.speed = value;
}

public void getSpeed() {
    return this.speed;
}
```